



Università degli Studi di Bologna
Facoltà di Ingegneria

Principles, Models,
and Applications
for Distributed Systems M

tutor

- Isam M. Al Jawarneh, PhD student
isam.aljawarneh3@unibo.it

Mobile Middleware Research Group – DISI –
UNIBO – ITALY

Teaching method and time/location

- Hands-on using JAVA
- every Tuesday, from 11:00 to 13:00 A.M., LAB 2, ground floor

Topics

JAVA basics

Exception Handling

Multithreading

Networking

EXAM DATES

June

- 09/06/2018 at 11:00 a.m. in LAB2

July

- 07/07/2016 at 11:00 a.m. in LAB2

September

- On request

TEXT BOOKS

- Deitel, H. & Deitel, P. (2007). JAVA How to Program. Prentice Hall, USA
- B.J. Evans, D. Flanagan: "Java in a Nutshell - A Desktop Quick Reference", 6th edition, O'Reilly, 2014.

other RESOURCES

- http://www.tutorialspoint.com/java/java_quick_guide.htm

Java

- Object Oriented
 - Objects
 - Components
 - Classes
 - Methods
 - Graphical User Interface (GUI)
 - Event Driven
- Java Application Programming Interface (API)
- Platform Independent

Typical Java Development Environment

- Java programs go through five phases
 - Edit
 - Programmer writes program using an editor; stores program on disk with the .java file name extension
 - Compile
 - Use javac (the Java compiler) to create bytecodes from source code program; bytecodes stored in .class files
 - Load
 - Class loader reads bytecodes from .class files into memory
 - Verify
 - Bytecode verifier examines bytecodes to ensure that they are valid and do not violate security restrictions
 - Execute
 - Java Virtual Machine (JVM) uses a combination of interpretation and just-in-time compilation to translate bytecodes into machine language

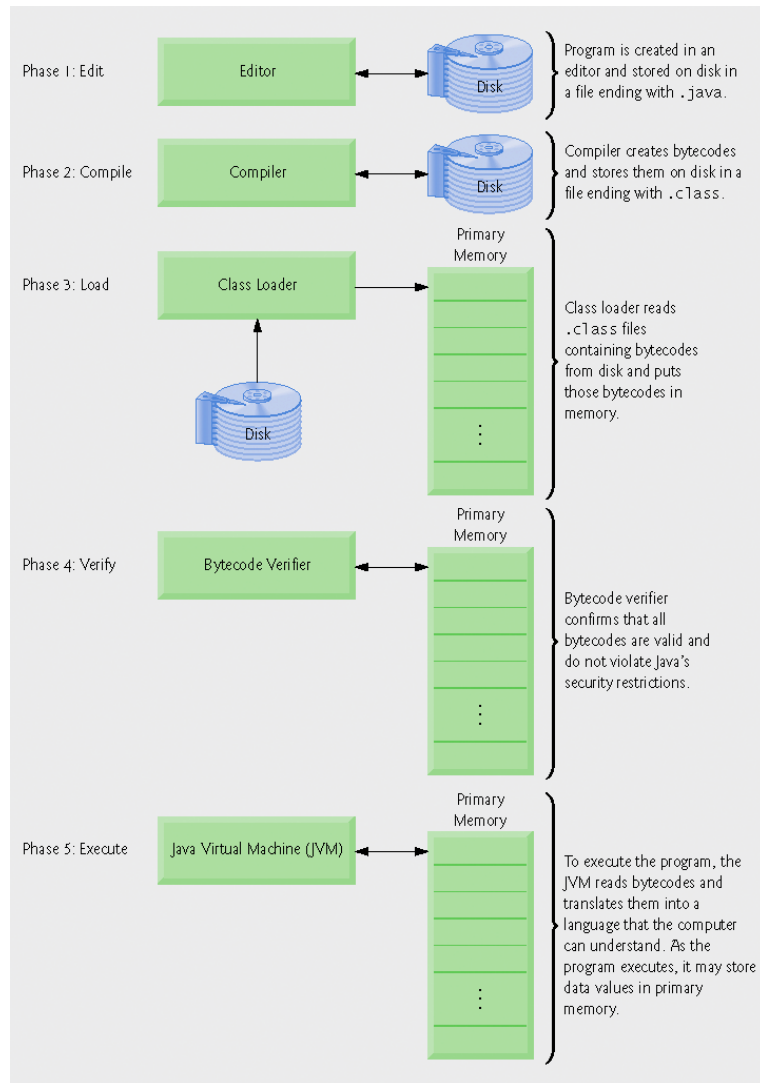
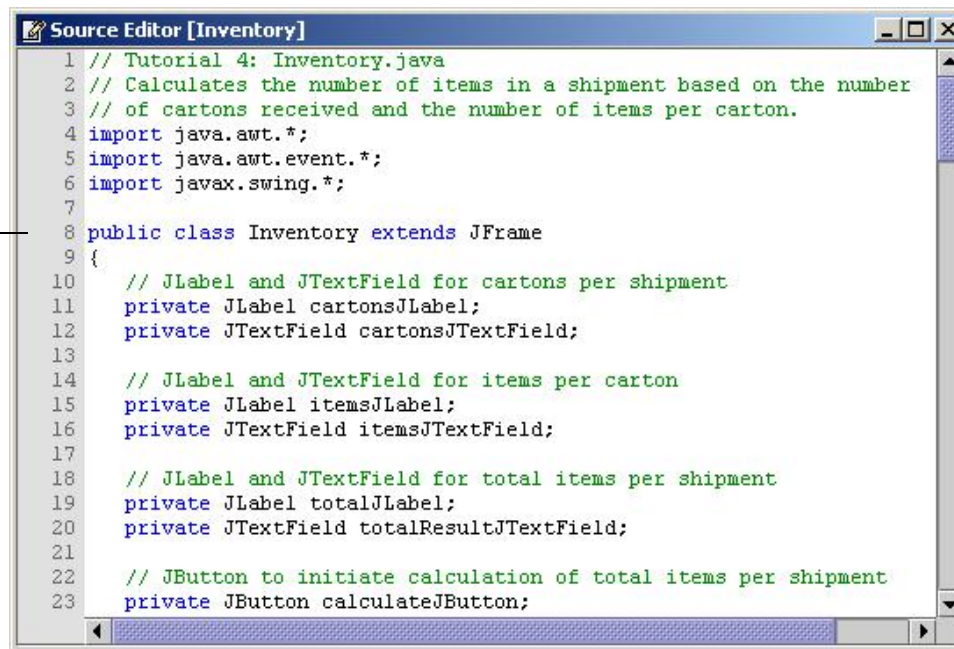


Fig. 1.1 | Typical Java development environment.

Introduction to Java Code

Beginning of
class declaration



```
1 // Tutorial 4: Inventory.java
2 // Calculates the number of items in a shipment based on the number
3 // of cartons received and the number of items per carton.
4 import java.awt.*;
5 import java.awt.event.*;
6 import javax.swing.*;
7
8 public class Inventory extends JFrame
9 {
10     // JLabel and JTextField for cartons per shipment
11     private JLabel cartonsJLabel;
12     private JTextField cartonsJTextField;
13
14     // JLabel and JTextField for items per carton
15     private JLabel itemsJLabel;
16     private JTextField itemsJTextField;
17
18     // JLabel and JTextField for total items per shipment
19     private JLabel totalJLabel;
20     private JTextField totalResultJTextField;
21
22     // JButton to initiate calculation of total items per shipment
23     private JButton calculateJButton;
```

Figure : Text editor showing a portion of the code for the Inventory application.

Introduction to Java Code (cont.)

- Java Code
 - Classes (Case sensitive)
 - Class declaration
 - Class keyword
 - Class name
 - Identifier
 - Left brace
 - Body
 - Right brace
 - Inherits
 - Extends
 - Methods
 - Blocks
 - Keywords (reserved words)

example

```
1 // Fig. 2.1: welcome1.java
2 // Text-printing program.
3
4 public class welcome1
5 {
6     // main method begins execution of Java application
7     public static void main( String args[] )
8     {
9         System.out.println( "welcome to Java Programming!" );
10
11     } // end method main
12
13 } // end class welcome1
```

```
Welcome to Java Programming!
```

- Welcome1.java

First Program in Java: Printing a Line of Text (Cont.)

```
1 // Fig. 2.1: welcome1.java
```

- Comments start with: `//`
 - Comments ignored during program execution
 - Document and describe code
 - Provides code readability

- Traditional comments: `/* ... */`

```
/* This is a traditional  
comment. It can be  
split over many lines */
```

```
2 // Text-printing program.
```

- Another line of comments
- Note: line numbers not part of program, added for reference

First Program in Java: Printing a Line of Text (Cont.)

```
4 public class welcome1
```

- Java identifier
 - Series of characters consisting of letters, digits, underscores (_) and dollar signs (\$)
 - Does not begin with a digit, has no spaces
 - Examples: Welcome1, \$value, _value, button7
 - 7button is invalid
 - Java is case sensitive (capitalization matters)
 - a1 and A1 are different

First Program in Java: Printing a Line of Text (Cont.)

```
4 public class Welcome1
```

- Saving files
 - File name must be class name with .java extension
 - Welcome1.java

```
5 {
```

- Left brace {
 - Begins body of every class
 - Right brace ends declarations (line 13)

First Program in Java: Printing a Line of Text (Cont.)

```
7 public static void main( String args[] )
```

- Part of every Java application
 - Applications begin executing at main
 - Parentheses indicate main is a method (Ch. 3 and 6)
 - Java applications contain one or more methods
 - Exactly one method must be called main
- Methods can perform tasks and return information
 - void means main returns no information
 - For now, mimic main's first line

```
8     {
```

- Left brace begins body of method declaration
 - Ended by right brace } (line 11)

First Program in Java: Printing a Line of Text (Cont.)

9

```
system.out.println( "welcome to Java Programming!" );
```

- Instructs computer to perform an action
 - Prints string of characters
 - String – series of characters inside double quotes
 - White-spaces in strings are not ignored by compiler
- System.out
 - Standard output object
 - Print to command window (i.e., MS-DOS prompt)
- Method System.out.println
 - Displays line of text
- This line known as a statement
 - Statements must end with semicolon ;

First Program in Java: Printing a Line of Text (Cont.)

```
11     } // end method main
```

- Ends method declaration

```
13 } // end class welcome1
```

- Ends class declaration
- Can add comments to keep track of ending braces

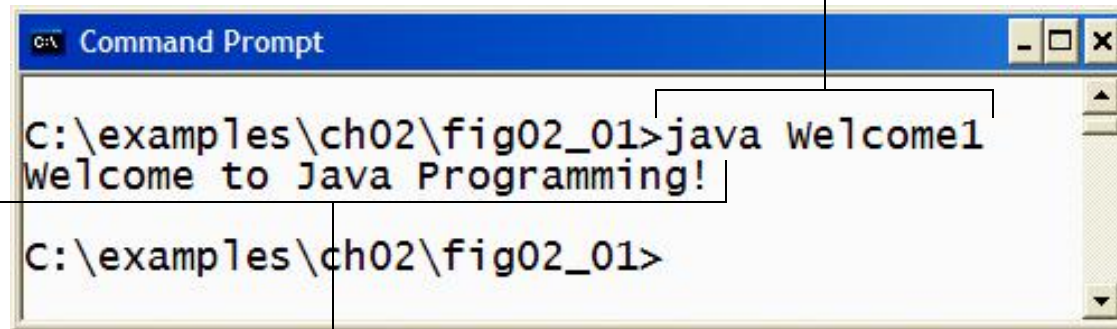
First Program in Java: Printing a Line of Text (Cont.)

- Compiling a program
 - Open a command prompt window, go to directory where program is stored
 - Type `javac Welcome1.java`
 - If no syntax errors, `Welcome1.class` created
 - Has bytecodes that represent application
 - Bytecodes passed to JVM

First Program in Java: Printing a Line of Text (Cont.)

- Executing a program
 - Type `java Welcome1`
 - Launches JVM
 - JVM loads .class file for class `Welcome1`
 - .class extension omitted from command
 - JVM calls method `main`

You type this command to
execute
the application



```
C:\examples\ch02\fig02_01>java Welcome1
Welcome to Java Programming!
C:\examples\ch02\fig02_01>
```

The program outputs

Welcome to Java Programming!

Fig. Executing Welcome1 in a Microsoft Windows Command Prompt window.

Outline

```
1 // Fig. 2.3: Welcome2.java
2 // Printing a line of text with multiple statements.
3
4 public class Welcome2
5 {
6     // main method begins execution of Java application
7     public static void main( String args[] )
8     {
9         System.out.print( "Welcome to " );
10        System.out.println( "Java Programming!" );
11    }
12 } // end method main
13
14 } // end class Welcome2
```

`System.out.print` keeps the cursor on the same line, so `System.out.println` continues on the same line.

```
Welcome to Java Programming!
```

Modifying Our First Java Program (Cont.)

- Escape characters
 - Backslash (\)
 - Indicates special characters to be output
- Newline characters (\n)
 - Interpreted as “special characters” by methods `System.out.print` and `System.out.println`
 - Indicates cursor should be at the beginning of the next line
 - `Welcome3.java` (Fig. 2.4)

```
9      System.out.println( "Welcome\nto\nJava\nProgramming!" );
```

- Line breaks at \n

Outline

```
1 // Fig. 2.4: Welcome3.java
2 // Printing multiple lines of text with a single statement.
3
4 public class Welcome3
5 {
6     // main method begins execution of Java application
7     public static void main( String args[] )
8     {
9         System.out.println( "welcome\n\tto\n\tJava\n\tProgramming!" );
10    } // end method main
11 } // end class Welcome3
```

```
Welcome
to
Java
Programming!
```

A new line begins after each `\n` escape sequence is output.

Escape sequence	Description
<code>\n</code>	Newline. Position the screen cursor at the beginning of the next line.
<code>\t</code>	Horizontal tab. Move the screen cursor to the next tab stop.
<code>\r</code>	Carriage return. Position the screen cursor at the beginning of the current line—do not advance to the next line. Any characters output after the carriage return overwrite the characters previously output on that line.
<code>\\</code>	Backslash. Used to print a backslash character.
<code>\"</code>	Double quote. Used to print a double-quote character. For example, <pre>System.out.println("\"in quotes\"");</pre> displays <pre>"in quotes"</pre>

Fig. | Some common escape sequences.

Displaying Text with printf

- `System.out.printf`
 - Feature added in Java SE 5.0
 - Displays formatted data

```
9      System.out.printf( "%s\n%s\n",  
10         "welcome to", "Java Programming!" );
```

- Format string
 - Fixed text
 - Format specifier – placeholder for a value
- Format specifier `%S` – placeholder for a string

Outline

```
1 // Fig. 2.6: Welcome4.java
2 // Printing multiple lines in a dialog box.
3
4 public class Welcome4
5 {
6     // main method begins execution of Java application
7     public static void main( String args[] )
8     {
9         System.out.printf( "%s\n%s\n",
10            "welcome to", "Java Programming!" );
11
12     } // end method main
13
14 } // end class Welcome4
```

System.out.printf
displays formatted data.

```
Welcome to
Java Programming!
```

Another Java Application: Adding Integers

- Upcoming program
 - Use `Scanner` to read two integers from user
 - Use `printf` to display sum of the two values
 - Use packages

Outline

```

1 // Fig. 2.7: Addition.java
2 // Addition program that displays the sum of two numbers.
3 import java.util.Scanner; // program uses class Scanner
4
5 public class Addition
6 {
7     // main method begins execution of Java application
8     public static void main( String args[] )
9     {
10        // create Scanner to obtain input from command window
11        Scanner input = new Scanner( System.in );
12
13        int number1; // first number to add
14        int number2; // second number to add
15        int sum; // sum of number1 and number2
16
17        System.out.print( "Enter first integer: " ); // prompt
18        number1 = input.nextInt(); // read first number from user
19

```

import declaration imports class Scanner from package java.util.

java

Declare and initialize variable input, which is a Scanner.

Declare variables number1, number2 and sum.

Read an integer from the user and assign it to number1.

Outline

```

20  System.out.print( "Enter second integer: " ); // prompt
21  number2 = input.nextInt(); // read second number from user
22
23  sum = number1 + number2; // add numbers
24
25  system.out.printf( "Sum is %d\n", sum ); // d
26
27  } // end method main
28
29 } // end class Addition

```

Read an integer from the user and assign it to

Calculate the sum of the variables `number1` and `number2`, assign result to

Display the sum using formatted output.

```

Enter first integer: 45
Enter second integer: 72
Sum is 117

```

Two integers entered by the user.

- Addition.java
- (2 of 2)
- 4. Addition
- 5. printf

Another Java Application: Adding Integers (Cont.)

```
3 import java.util.Scanner; // program uses class Scanner
```

- `import` declarations
 - Used by compiler to identify and locate classes used in Java programs
 - Tells compiler to load class `Scanner` from `java.util` package

```
5 public class Addition  
6 {
```

- Begins `public class Addition`
 - Recall that file name must be `Addition.java`
- Lines 8-9: begin `main`

Another Java Application: Adding Integers (Cont.)

```
10 // create Scanner to obtain input from command window
11 Scanner input = new Scanner( System.in );
```

- Variable Declaration Statement
- Variables
 - Location in memory that stores a value
 - Declare with name and type before use
 - Input is of type Scanner
 - Enables a program to read data for use
 - Variable name: any valid identifier
- Declarations end with semicolons ;
- Initialize variable in its declaration
 - Equal sign
 - Standard input object
 - `System.in`

Another Java Application: Adding Integers (Cont.)

```
13     int number1; // first number to add
14     int number2; // second number to add
15     int sum; // sum of number 1 and number 2
```

- Declare variable `number1`, `number2` and `sum` of type `int`
 - `int` holds integer values (whole numbers): i.e., 0, -4, 97
 - Types `float` and `double` can hold decimal numbers
 - Type `char` can hold a single character: i.e., x, \$, \n, 7
 - `int`, `float`, `double` and `char` are primitive types
- Can add comments to describe purpose of variables

```
int number1, // first number to add
    number2, // second number to add
    sum; // sum of number1 and number2
```

- Can declare multiple variables of the same type in one declaration
- Use comma-separated list

Another Java Application: Adding Integers (Cont.)

```
13     int number1; // first number to add
14     int number2; // second number to add
15     int sum; // sum of number 1 and number 2
```

- Declare variable `number1`, `number2` and `sum` of type `int`
 - `int` holds integer values (whole numbers): i.e., 0, -4, 97
 - Types `float` and `double` can hold decimal numbers
 - Type `char` can hold a single character: i.e., x, \$, \n, 7
 - `int`, `float`, `double` and `char` are primitive types
- Can add comments to describe purpose of variables

```
int number1, // first number to add
    number2, // second number to add
    sum; // sum of number1 and number2
```

- Can declare multiple variables of the same type in one declaration
- Use comma-separated list

Another Java Application: Adding Integers (Cont.)

```
20      System.out.print( "Enter second integer: " ); // prompt
```

- Similar to previous statement
 - Prompts the user to input the second integer

```
21      number2 = input.nextInt(); // read second number from user
```

- Similar to previous statement
 - Assign variable number2 to second integer input

```
23      sum = number1 + number2; // add numbers
```

- Assignment statement
 - Calculates sum of number1 and number2 (right hand side)
 - Uses assignment operator = to assign result to variable sum
 - Read as: sum gets the value of number1 + number2
 - number1 and number2 are operands

Another Java Application: Adding Integers (Cont.)

```
25      System.out.printf( "sum is %d\n: " , sum ); // display sum
```

- Use `System.out.printf` to display results
- Format specifier `%d`
 - Placeholder for an `int` value

```
      System.out.printf( "Sum is %d\n: " , ( number1 + number2 ) );
```

- Calculations can also be performed inside `printf`
- Parentheses around the expression `number1 + number2` are not required

Memory Concepts

- Variables
 - Every variable has a name, a type, a size and a value
 - Name corresponds to location in memory
 - When new value is placed into a variable, replaces (and destroys) previous value
 - Reading variables from memory does not change them



Fig. | Memory location showing the name and value of variable **number1**.

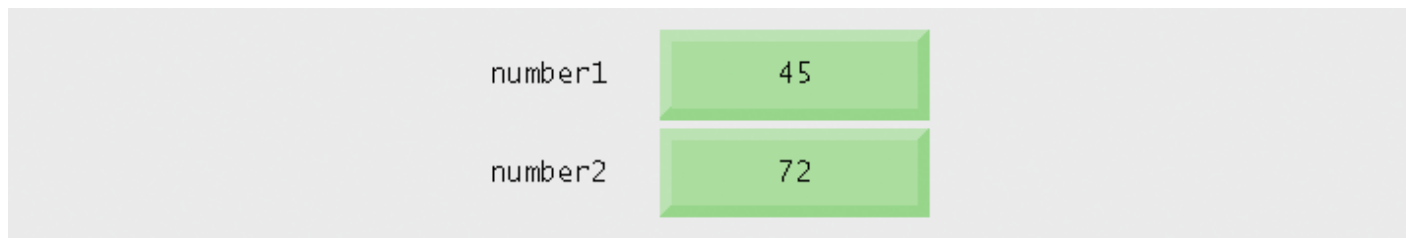


Fig. | Memory locations after storing values for **number1** and **number2**.

number1	45
number2	72
sum	117

Fig. | Memory locations after calculating and storing the sum of **number1** and **number2**.

Arithmetic

- Arithmetic calculations used in most programs
 - Usage
 - * for multiplication
 - / for division
 - % for remainder
 - +, -
 - Integer division truncates remainder
 - 7 / 5 evaluates to 1
 - Remainder operator % returns the remainder
 - 7 % 5 evaluates to 2

Java operation	Arithmetic operator	Algebraic expression	Java expression
Addition	+	$f+7$	<code>f + 7</code>
Subtraction	-	$p-c$	<code>p - c</code>
Multiplication	*	bm	<code>b * m</code>
Division	/	x/y or $\frac{x}{y}$ or $x \div y$	<code>x / y</code>

Fig. | Arithmetic operators.

Arithmetic (Cont.)

- Operator precedence
 - Some arithmetic operators act before others (i.e., multiplication before addition)
 - Use parenthesis when needed
 - Example: Find the average of three variables a , b and c
 - Do not use: $a + b + c / 3$
 - Use: $(a + b + c) / 3$

Operator(s)	Operation(s)	Order of evaluation (precedence)
*	Multiplication	Evaluated first. If there are several operators of this type, they are evaluated from left to right.
/	Division	
%	Remainder	
+	Addition	Evaluated next. If there are several operators of this type, they are evaluated from left to right.
-	Subtraction	

Fig. | Precedence of arithmetic operators.

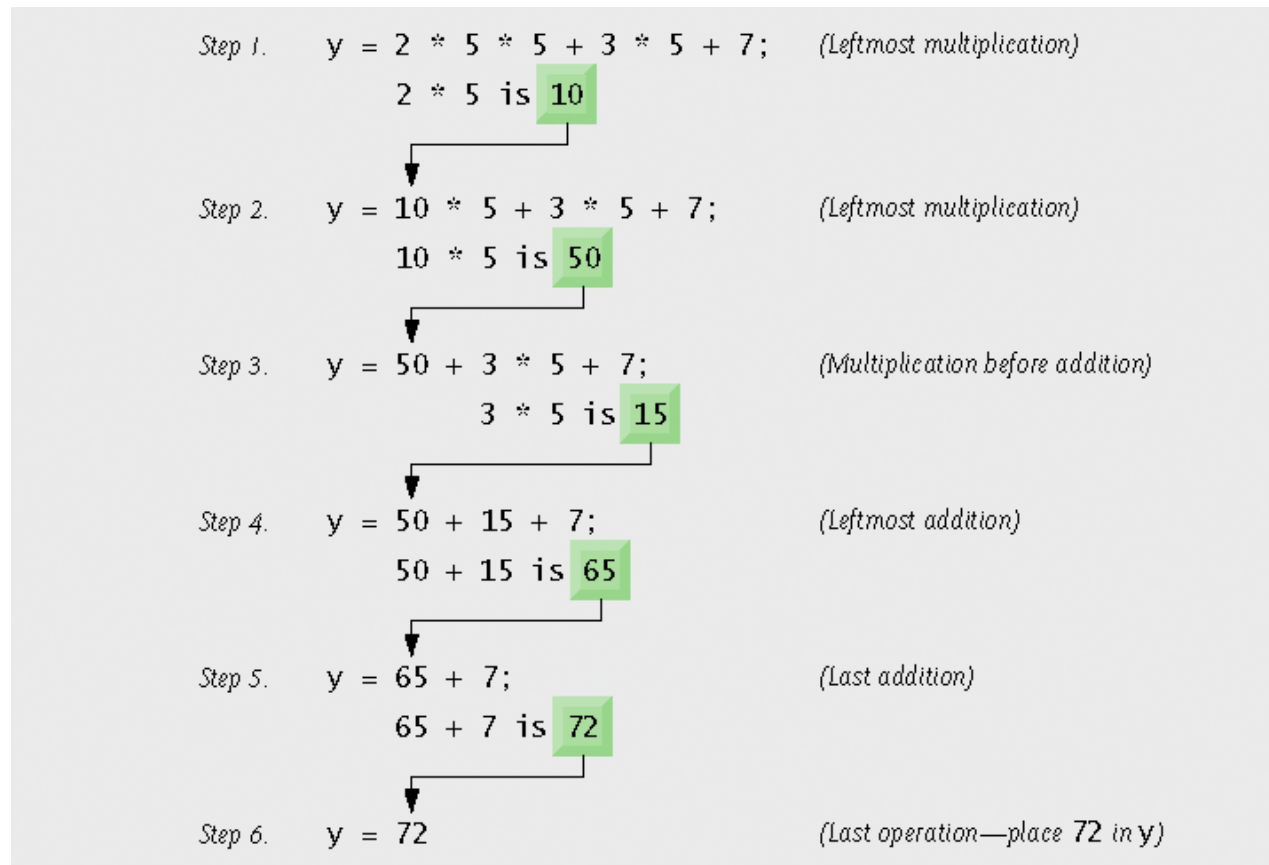


Fig. | Order in which a second-degree polynomial is evaluated.

Decision Making: Equality and Relational Operators

- Condition
 - Expression can be either `true` or `false`
- `if` statement
 - Simple version in this section, more detail later
 - If a condition is `true`, then the body of the `if` statement executed
 - Control always resumes after the `if` statement
 - Conditions in `if` statements can be formed using equality or relational operators (next slide)

Standard algebraic equality or relational operator	Java equality or relational operator	Sample Java condition	Meaning of Java condition
<i>Equality operators</i>			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
<i>Relational operators</i>			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Fig. 2.14 | Equality and relational operators.


```

1 // Fig. 2.15: Comparison.java
2 // Compare integers using if statements, relational operators
3 // and equality operators.
4 import java.util.Scanner; // program uses class Scanner
5
6 public class Comparison
7 {
8     // main method begins execution of Java application
9     public static void main( String args[] )
10    {
11        // create Scanner to obtain input from command window
12        Scanner input = new Scanner( System.in );
13
14        int number1; // first number to compare
15        int number2; // second number to compare
16
17        System.out.print( "Enter first integer: " ); // prompt
18        number1 = input.nextInt(); // read first number from user
19
20        System.out.print( "Enter second integer: " ); // prompt
21        number2 = input.nextInt(); // read second number from user
22
23        if ( number1 == number2 )
24            System.out.printf( "%d == %d\n", number1, number2 );
25
26        if ( number1 != number2 )
27            System.out.printf( "%d != %d\n", number1, number2 );
28
29        if ( number1 < number2 )
30            System.out.printf( "%d < %d\n", number1, number2 );

```

Test for equality,
display result using
printf.

Compares two numbers
using relational operator
<.

Outline

• Comparison.java

- (1 of 2)
- 1. Class Comparison
 - 1.1 main
 - 1.2 Declarations
 - 1.3 Input data (nextInt)
 - 1.4 Compare two inputs using if statements

```

31
32     if ( number1 > number2 )
33         System.out.printf( "%d > %d\n", number1, number2 );
34
35     if ( number1 <= number2 )
36         System.out.printf( "%d <= %d\n", number1,
37
38     if ( number1 >= number2 )
39         System.out.printf( "%d >= %d\n", number1, number2 );
40
41 } // end method main
42
43 } // end class Comparison

```

Compares two numbers using relational operators >, <= and >=.

```

Enter first integer: 777
Enter second integer: 777
777 == 777
777 <= 777
777 >= 777

```

```

Enter first integer: 1000
Enter second integer: 2000
1000 != 2000
1000 < 2000
1000 <= 2000

```

```

Enter first integer: 2000
Enter second integer: 1000
2000 != 1000
2000 > 1000
2000 >= 1000

```

Outline

- Comparison.java
 - (2 of 2)
- Program output

Decision Making: Equality and Relational Operators (Cont.)

- Line 6: begins class `Comparison` declaration
- Line 12: declares Scanner variable `input` and assigns it a Scanner that inputs data from the standard input
- Lines 14-15: declare `int` variables
- Lines 17-18: prompt the user to enter the first integer and input the value
- Lines 20-21: prompt the user to enter the second integer and input the value

Decision Making: Equality and Relational Operators (Cont.)

```
23     if ( number1 == number2 )
24         System.out.printf( "%d == %d\n", number1, number2 );
```

- `if` statement to test for equality using (`==`)
 - If variables equal (condition true)
 - Line 24 executes
 - If variables not equal, statement skipped
 - No semicolon at the end of line 23
 - Empty statement
 - No task is performed
- Lines 26-27, 29-30, 32-33, 35-36 and 38-39
 - Compare `number1` and `number2` with the operators `!=`, `<`, `>`, `<=` and `>=`, respectively

Operators				Associativity	Type
*	/	%		left to right	multiplicative
+	-			left to right	additive
<	<=	>	>=	left to right	relational
==	!=			left to right	equality
=				right to left	assignment

Fig. 2.16 | Precedence and associativity of operations discussed.

References

- Deitel, H. & Deitel, P. (2007). JAVA How to Program. Prentice Hall, USA